大容量データベースの圧縮利用法
日経財務データの圧縮とパソコンでの利用

内　藤　三　義

はじめに

昨今，コンピュータ用外部記憶装置のコンパクト化とアクセス・スピードの向上は，ハード装置の価格の低下とあいまって，コンピュータの利用環境に大きな変化をもたらしている。かつては，メガバイトという単位を越えるようなファイル（データ・ベース）を操作するのは，ほとんど大型計算機の世界に限られていたのが，パーソナル・コンピュータで20～80メガ・バイトのハードディスクを搭載するのが通常のこととなった現在では，かなりの容量のファイルも，ユーザーの利用目的によっては，直接パーソナル・コンピュータで処理できるようになってきている。しかし，コンピュータのハード能力の向上は，同時に，ユーザーが利用したい情報量をも飛躍的に増大させているので，より大きなデータをより高速で扱えるハードへの改良や，大容量データを合理的に圧縮コンパクト化するソフトへのニーズをも高めている。

そこで本稿では，日本経済新聞社が提供している，「NEEDS　日経財務データ」のうち，「一般事業会社・本決算データ（磁気テープ）」（以下，NEEDS データと略す）を対象にしたデータ圧縮法と，パーソナル・コンピュータでの利用法について，筆者の試みを紹介する[1)]。

1．NEEDS データの構成と圧縮の目的

NEEDS データは，全国証券取引所上場会社（約1800社）の本決算データなどの411項目を1964年度以降収録しているもので，1988年 6 月決算期迄の累計で，57005レコード，約170メ

ガ・バイトの容量となっている。このようにかなり大容量のため，このデータを利用しようとするユーザーは，大型計算機センターなどを利用してこの磁気テープ・データを企業コードなどからアクセスが可能な VSAM 形式ファイルに変換してディスクに保存し，そのファイル形式に見合った出力プログラムを各自作成することで必要なデータを取り出す，という方法で使うのが一般的であったようである。しかし，最近はパーソナル・コンピュータ・ユーザーが増大し，出力データをパソコンで再処理したいというニーズが高まっており，またそのことでより効果的な解析を得られることも多くなった。その場合，(1) 大型計算機の出力ファイルをパソコン用フロッピー・ディスクにおとして利用する方法，(2) パソコンから通信で大型計算機を利用し，出力を直接パソコン・ディスクに転送する方法，などがこれまでは考えられたが，大型計算機センターの利用時間の制限や，特に，出力をパソコン・ディスクに転送する作業の煩わしさと，処理時間の遅さを考えると，最初からパソコンにデータ・ベースを設定して，出力が直接パソコンで得られれば非常に便利なのは間違いないところである。

企業財務データを対象にした計量的な研究では，データを取り出すのが１度で終ることは少ない。ある仮説で以て取り出したデータ群の解析の結果，さらに別のデータ群を取り出して解析することが必要になることはめずらしいことではない。このような時に，その処理が連続的に１つのマシン上で行なうことが出来れば，計量的解析の一層の深化に繋がることも大いにあるのではないだろうか。しかしパソコンでこれを行なおうとして，約170メガ・バイトというオリジナルのデータ量をそのまま移植するのは，ハード・ディスクの大容量化の現状でも容易なことではない。パソコンで，そしてそのハード・ディスクにデータを常駐させて利用出来るようにするためには，10メガあるいは20メガ・バイトという単位以下にデータを整理するのか圧縮することがどうしても必要である。

そこで，第１目標として20メガ・バイト程度にデータを圧縮しようとしたのであるが，通常のデータ圧縮技法の利用では，オリジナル・データ量で1/8以下にするというような圧縮を達成するのは困難であった。この NEEDS データの特性にうまく対応した圧縮法を用いる必要があることが想定されたが，以下に説明するような独自な方法を利用することで概ね成功したので，その方法を順に説明する。

1）この論文の基礎となっているものは，大阪市立大学経済学部の大川勉教授と，同学部堀山秀一助教授らと共に進めた企業分析研究会での総合経済・経営分析システムを作成する研究作業である。同研究会では，研究者がパソコンでも容易に一般事業会社決算データのような大容量データ・ベースにアクセスできることをめざし，同データの CD—ROM 化を達成した（「日経パソコン」86年10月20日号参照）。しかし，個々の大学・学部・研究者が購入しているデータを，個々に，かつ継続的に CD—ROM 化するのは，時間の点でも費用の点でも非常なコストを伴うものである。そこでより容易にパソコン用のデータ・ベースとして利用できる方法を検討し，この論文で示すような方法を開発した。この研究・開発にあたっては，大川・堀山の両先生には継続的に非常に多くの示唆を頂いている。記して謝意を表す次第である。

2．NEEDS データ圧縮法

NEEDS データは，既に説明したように1964年以降継続的に更新される411項目のデータからなるが，磁気テープ媒体で提供されているファイルの形式と仕様は次の様になっている。

ファイル形式[2)]

編　成　Sequential File

Block Size　15,000 bytes

Record Size　3,000 bytes（固定長方式）

入力書式　：　固定書式

データ形式

数　値　Packed Decimal[3)]　9桁（5 bytes）

文字列　EBCDIC コード[4)]　（1文字1 byte）

1社の1決算期のデータは，1レコード3,000 bytes に収められているが，このうち大部分は Packed Decimal 9桁の数値データであり，これが副次項目を含めると415項目あり，1レコードの約3分の2にあたる2,075 bytes を占める。残りの925 bytes は文字，または文字列のデータ項目であるが，内39 bytes 分は，EBCDIC コードの数字1～5カラムからなるものであり，入力は文字（列）であるが，数値としても扱えるデータとなっている。従ってNEEDS データの圧縮のためには，なによりもこの数値データ，特に10進数9桁の数値を効率良く圧縮することが必要である。

2）Sequential File とは，1つのまとまりを持ったデータ単位である Record が，何等かの順番に収められているファイルということであり，データの入出力は通常第1レコードから昇順に行う。Record Size はこのまとまりのあるデータ単位が何バイトから構成されているかということであり，このサイズが全てのレコードについて同一である固定長と，レコード毎に大きさの異なる可変長などの種類がある。なお，固定書式とは1レコードの何バイト目から何バイトはどのようなデータがどのような記憶様式で収められているか前もって決まって書き込まれているファイルということであり，固定書式ファイルは多くの場合固定長ファイルとなる。

3）Packed Decimal とは，コンピュータで数字を扱うときの形式の1つで，10進数の数字1桁を0.5 byte（4 bits の0000_b～1001_b）で表現するものである。数値の正負も同じく4 bits で表され通常1011_bか1111_bがあてられる。Packed Decimal の9桁は，従って0.5×9桁＋0.5（正負の判定用）＝5 bytes となる。

4）コンピュータでは通常，英数カナなどの文字（半角文字）1文字を1 byte で表現するが，その時，1 byte で表現できる0～255までのコードのどの番号にどの文字を対応させるかを前もって決めておく必要がある。この対応のさせかたの1つが EBCDIC コードと呼ばれるもので，コード番号の64～255に通常文字をあてはめ，0～63にはコンピュータを制御する様々な特殊コードをおいている。汎用計算機で使用する文字コードは概ねこのコードであるが，パソコンでは32～255に通常文字を対応させて使う ASCII コードを主に利用している。

2-1．コンピュータでの数値の扱い

コンピュータで数値を扱う場合，通常は数値を2進数で表現し，整数は2バイト，4バイト等で，また浮動小数点は，4バイト，8バイトなどのようにコンピュータで扱い易い，16, 32, 64 bits 単位にまとめて記録，処理している。いま，整数だけを考えるのなら，x bits からなる2進数 Bx で表すことが出来る10進数の最小値と最大値は，マイナスの値を2の補数で表すなら，

$$-2^{x-1} \leqq Bx \leqq 2^{x-1}-1$$

ex) 8 bits なら $-2^{x-1} = -128 \leqq B_8 \leqq 127 = 2^{x-1}-1$

となるので，

10進数1桁（－9～9）：2進数5桁（5 bits）

10進数2桁（－99～99）：2進数8桁（1 byte）

・・・・・・・・・・・・・・・・

10進数9桁　　：2進数31桁（3 bytes ＋7 bits）

で表すことができる。別の言い方をすると，Packed Decimal の9桁，5バイトの整数（固定小数点形式の数値も小数点以下が何桁か指定されていれば同じである）は，2進数表記法による4バイト整数で数値の精度の変更無く置き換えれると言える。この方法だけで，Packed Decimal の9桁の数値データは，5分の4に圧縮できるわけであるが，それだけでは圧縮率があまりに低い。そこで，NEEDS データに実際に収録されている数値をみると，入力されているデータのすべてが9桁の数値であるのではなく，実際は，それ以下の桁数で表現されている数値が多いこと，また，データ項目の入力が無い欠損値と指定されている－999999999が非常に多いことが分かった。それならば，実際の数値の桁数に見合った領域を，各項目の数値に割り当てれば，データ圧縮率はかなり高くなる筈である。しかし，そのような圧縮をするためには，1つの項目の数値が占める領域（ビット数・バイト数）が前もって決ってこないので，数値と数値の間に，何か確実な区切りを設定するのか，それとも，1つ1つの数値が入力された長さが判定できるインデックスを設けることが必要となる。このうち，「区切り」をなんらかのコードで挿入することは，圧縮を実現しながらは実際は不可能であるので，インデックス・データに圧縮して入力した実際の数値桁数を表現させ，それに対応して数値を指定された長さだけ書き込むという措置が必要となる。では，インデックス・データを一体どのように設定し，そして実データをどのように表現させればよいのだろうか。その点については，整数である実データをバイト単位で表す圧縮法1，4ビット単位で表す圧縮法2，ビット単位で表す圧縮法3の3つの方式を以下の様に考えた。

2-2-1　圧縮法1（2ビット固定インデックス，0～4バイト可変長データ）

圧縮法1は実データを0～4バイトからなるバイト単位の2進数の整数で表そうとするものであるので，実データのバイト長は5種類あることになる。この5種類という区別を表すインデックスのコード化のためには，固定長なら3ビット｛0＝000_b，1＝001_b，‥4＝100_b，5＝101_b｝，可変長なら5ビットまで｛0＝1_b，1＝01_b，2＝001_b，3＝0001_b，4＝00001_b，5＝00000_b，など｝，のインデックス領域が必要なことになる。どちらの方式をとっても，3バイト以下で表現できる実データが多ければかなりの圧縮が期待できる。しかし，このバイト単位で実データを表すのは，圧縮データから4バイト整数への読み込みと変換が容易なことにも特徴があるので，インデックス長を固定化することで，また，出来ればそのインデックス・データもバイト単位で整理可能であることが，特徴を生かすことになる。そこで，可変長データを0，1，2，4バイト長の4種類，インデックスを固定長2ビットとしたのが表1で示す圧縮法1である。欠損値がほとんどないデータを対象にするなら表1を変形し，1～4バイトの4種類に00_b～11_bのインデックス・コードを与えればよいし，1バイトで表現できる整数の頻度が少ないデータが対象なら，0，2，3，4バイト整数の4種類を選択しても良い。

表1　圧縮法1のインデックスと圧縮データ長

実データの数値	index 1	圧縮データ長（byte）
−999999999	00_b	0
−128～127　＊	01_b	1
−32768～32767 ＊	10_b	2
上記以外の数値	11_b	4

※　＊のついた3段目以降の数値の領域は，それより上の段で示された数値領域を除くものとする（以下の表2～表3も同じ）

2-2-2　圧縮法2（1～8ビット可変長インデックス，
0～32ビット・4ビット単位可変長データ）

圧縮法1では圧縮データをバイト単位の可変長で表そうとしたが，圧縮法2（表2）では1/2バイト（4ビット）単位で表すことで，データ数値のレンジが広い（数値のバラツキが大きい）時に，より効率的な圧縮を図ろうとするものである。圧縮データ長の種類は，0～4バイトで9種類になるので，1種類をどれかに統合することで割愛しても，8種類の区別を表す3ビットの固定長インデックス，若しくは1～8ビットの可変長インデックスが必要となる。

なお，欠損値を無視するなら，表2の index 2-1の先頭1ビットは省略でき，index 2-2は0.5～4 bytes まで，すべて0.5 byte 単位で表現できる。また，index 2-1はこの上下の並びを変更することもできる。

表 2　圧縮法 2 のインデックスと圧縮データ長

実データの数値	index 2-1	index 2-2	圧縮データ長 byte
−999999999	1	000	0
−8〜7	01	001	0.5
−128〜127	001	010	1
−2048〜2047	0001	011	1.5
−32768〜32767	00001	100	2
−524288〜524287	000001	101	2.5
−8388608〜8388607	0000001	110	3
−134217728〜134217727	00000001	111	3.5 *
上記以外の数値	00000000	111	4

* index 2-2の時は4 bytes
index 2-1，index 2-2はビット表現

2-2-3　圧縮法 3（1〜9 ビット可変長インデックス，0〜31ビット可変長データ）

圧縮法 3 は，可変長の単位を，最小のビット単位にまで下げて，実データを表そうとするものであり，インデックスの形式をうまく整えれば，より高い圧縮率を実現出来ることも期待される。この場合，実データの先頭 1 ビットは必ず 1 であるので[5]，その 1 ビットは省略できる。なお，この形式では正負判定ビットが独立し，負の数値は正の値の補数ではなく，その絶対値で表現する。圧縮データの可変長の種類は 0 〜30ビットの31種類であるので，5 ビット固定長，もしくは 1 〜30ビットの可変長インデックスが必要となる。しかし NEEDS データの値のバラツキから考えると，単純な固定長もしくは可変長インデックスでは，圧縮率が高くならないので，両者を組み合わせる方式を考えた。

欠損値をインデックスで無視してよければ表 3 のインデックスの先頭 1 ビットが省略出来るのは 2-2-2 と同様であり，また index 3-1〜index3-3の並びは上下で変更出来る。

表3　圧縮法3のインデックスと圧縮データ長

実データの範囲	index 3-1	index 3-2	index 3-3	圧縮データビット長	正負判定ビット
−999999999	1	1	1	0	0
0	01x	01xx	01xxx	0	0
−1～1	01x	01xx	01xxx	0	1
−3～3	001x	01xx	01xxx	1	1
−7～7	001x	01xx	01xxx	2	1
−15～15	0001xx	001xx	01xxx	3	1
−31～31	0001xx	001xx	01xxx	4	1
−63～63	0001xx	001xx	01xxx	5	1
−127～127	0001xx	001xx	01xxx	6	1
−255～255	00001xxx	0001xxx	001xxx	7	1
−511～511	00001xxx	0001xxx	001xxx	8	1
−1023～1023	00001xxx	0001xxx	001xxx	9	1
−2047～2047	00001xxx	0001xxx	001xxx	10	1
$-2^{12}-1 \sim 2^{12}-1$	00001xxx	0001xxx	001xxx	11	1
$-2^{13}-1 \sim 2^{13}-1$	00001xxx	0001xxx	001xxx	12	1
$-2^{14}-1 \sim 2^{14}-1$	00001xxx	0001xxx	001xxx	13	1
$-2^{15}-1 \sim 2^{15}-1$	00001xxx	0001xxx	001xxx	14	1
$-2^{16}-1 \sim 2^{16}-1$	00000xxxx	0000xxxx	000xxxx	15	1
$-2^{17}-1 \sim 2^{17}-1$	00000xxxx	0000xxxx	000xxxx	16	1
$-2^{18}-1 \sim 2^{18}-1$	00000xxxx	0000xxxx	000xxxx	17	1
$-2^{19}-1 \sim 2^{19}-1$	00000xxxx	0000xxxx	000xxxx	18	1
$-2^{20}-1 \sim 2^{20}-1$	00000xxxx	0000xxxx	000xxxx	19	1
$-2^{21}-1 \sim 2^{21}-1$	00000xxxx	0000xxxx	000xxxx	20	1
$-2^{22}-1 \sim 2^{22}-1$	00000xxxx	0000xxxx	000xxxx	21	1
$-2^{23}-1 \sim 2^{23}-1$	00000xxxx	0000xxxx	000xxxx	22	1
$-2^{24}-1 \sim 2^{24}-1$	00000xxxx	0000xxxx	000xxxx	23	1
$-2^{25}-1 \sim 2^{25}-1$	00000xxxx	0000xxxx	000xxxx	24	1
$-2^{26}-1 \sim 2^{26}-1$	00000xxxx	0000xxxx	000xxxx	25	1
$-2^{27}-1 \sim 2^{27}-1$	00000xxxx	0000xxxx	000xxxx	26	1
$-2^{28}-1 \sim 2^{28}-1$	00000xxxx	0000xxxx	000xxxx	27	1
$-2^{29}-1 \sim 2^{29}-1$	00000xxxx	0000xxxx	000xxxx	28	1
$-2^{30}-1 \sim 2^{30}-1$	00000xxxx	0000xxxx	000xxxx	29	1
該当数値なし	00000xxxx	0000xxxx	000xxxx	−	−

index 3-1～3-3で数字0，1が記載されている所は可変長ビット，x の字が記載されている所は固定長ビットである

2-2-4　圧縮法の違いによる実際の圧縮率と圧縮法の選択

以上，9桁の数値（32ビットの固定小数点）を圧縮するいくつかの方法を説明したが，では実際にこれらの方式による NEEDS データの圧縮率はどのようになるのだろうか。オリジナル・データの収録会社数の1％にあたる20社を無作為抽出して，収録全決算期について，82年度以降も継続して，データ入力されている263項目の Packed Decimal 数値について圧縮率を計算した結果は表4のようになった。

表4 圧縮法1～3によるNEEDSデータの圧縮率（20社263項目を抽出）

	原データ	圧縮法1	圧縮法2		圧縮法3		
			index 2-1	index 2-2	index 3-1	index 3-2	index 3-3
bytes	765330	204207	190698	195473	198189	188048	181155
指数1	100.00	26.68	24.92	25.54	25.90	24.57	23.67
指数2	100.00	—	23.71	—	23.81	23.13	23.14

※ 指数1は，2-2-1～2-2-3で説明した方法でデータ圧縮したときの，オリジナル・データに対する圧縮率，指数2はインデックスの必要ビット数をオリジナル・データの出現頻度の多いものには少ないビットを割り当てることで，各インデックスの上下を並べかえたもの。

インデックスの特別な並べ代えをしない指数1，をみると，圧縮法3のindex 3-3が，最も高い圧縮率を実現でき，指数2までで比べてみるとindex 3-2とindex 3-3がほぼ同じ圧縮率の高さを実現できるようである。それでは，実際のNEEDSデータ圧縮はこのどれを利用すべきなのだろうか。単に今回テストしたデータの圧縮率を高めることだけを課題とするなら，並べ代えをしたindex 3-2若しくはindex 3-3の方式を用いるべきである。しかし，ここでは年々更新されることで，あるレンジの数値出現頻度が大きく変化するかもしれないようなデータの圧縮を考えるのであるから，毎年入力数値を点検しインデックスを並べ代えることで最大の圧縮率を達成していくという方式はあまり良策ではないように思える。かなりの期間変更する必要のない圧縮方法を考えることが必要である。とするならば，index 3-3の一般方式が実効的な圧縮方式と言えるかもしれない。

しかしどのような圧縮方式を採用するかは，継続的な圧縮率の高さだけから決められるものではない。データ圧縮処理が容易なこと，特に圧縮データを読み込むパソコン側のプログラミングと，処理スピードも検討せねばならない。そこで筆者は，当面は，圧縮率は表中では最も低いが，プログラミングが汎用計算機においても，パソコンにおいても最も容易で，かつパソコンの処理スピードが最も速くなると思われる，圧縮法1を当面は採用した[5]。インデックス・データを実データと切り離した別個のファイルとして設定するとき，インデックスの長さが固定し，固定長のファイルになっていた方がパソコンの操作で便利であることも理由の1つになっている。

2-2-5 文字と文字列の圧縮

NEEDSにはPacked Decimalの数値以外に，文字と文字列が入力されているが，EBCDICコードの数字だけが入力されている箇所も多い。これは形式は文字であるが数値と

5）NEEDSデータでは，150項目程度のPacked Decimal数値データ領域が，将来のデータ入力用の領域等を確保するため空に（欠損値）してある。この領域も圧縮データで確保するには，圧縮法1では150×2 bits×57005 records＝2.14 MBをインデックス・ファイルの中に余分に確保せねばならないが，NEEDSデータの入力項目の変更は，それほど頻繁におこらないので，今回の作業ではその領域を割愛した。しかし，厳密な将来への対応を考えるならば，この領域を確保しておくことが必要であるので，一定時期になれば，この圧縮法1ではなく，欠損値データのためのインデックス領域が1 bitでよい圧縮法3（index 3-3形式）に移行することが望ましいと言える。

しても扱えるので，それらは「数値」の桁数に応じて，1～16 bits の固定長でデータを扱うことにした。英文とカナが混じる文字列からなるデータ項目としては，会社名（当該期のものと，変更があれば変更前のもの）が2箇所にそれぞれ35 bytes と25 bytes で入力されている。これらについては，最新の決算期に表示されている会社名を，社名情報ファイルの中に書き出し，変更前社名は，実際の社名長を1バイト整数でインデックスとし，実データ部分には，その長さに見合った ASCII コード文字列として入力することにした。文字列の圧縮にはもっと効果的な方法があるが，社名変更はそう度々ないので，ほとんどが実質データ長が0となるので，このような方法をとることとした。

3．圧縮データ・ベースの構成

以上のような圧縮法をとったので，パソコン用圧縮データ・ベースは単一ファイルではなく，次のような3つのファイルに編成した。

(1) 基本データファイル　：　実データの大部分を保存するファイルで，1社1期分のレコードは固定されていないが，1社のデータは最古期から最新期まで連続している。ファイル容量は14,611 KB

(2) インデックス・ファイル　：　実データのインデックスを(1)の並びの順に保存するファイルで，1社1期分のレコード長は固定されている（75 bytes)。また，当該データの決算期情報は(1)ではなくここに収納した。また，実データのレコード長も記入した。ファイル容量 3,953 KB

(3) 会社情報ファイル　：　最新決算期の英文・カナ社名，日経会社コード，株式コード，業種コード，上場情報などを収録し，これらは基本データから割愛した。また NEEDS データにはない日本語社名を入力した。ここには，収録開始・最新決算期，収録決算数，(1)で当該会社のデータが入力されている開始ポインタと入力総バイト数，(2)のインデックス開始ポインタなどを収めた。1社については1レコードだけが，固定長（128 byte）で保存されている。ファイル容量 256 KB

3つのファイルに分けたのは，圧縮率を高めるという目的もあったが，同時に，階層型あるいはリレーショナル型のデータ・ベースとしてのデータ・アクセスを容易しようとしたこと，更に，3つに分割することでそれぞれを異なったディスクに置くこともできるので，1つのディスクへの負荷を小さくできるメリットがあるからである。このようにしたので，パソコンでデータを読み込みなんらかの出力を得ようとする時は，企業情報ファイル→インデックス・ファイル→基本データ・ファイルと順に読み込み，変換することで必要なデータ項目数値や文字列を得るという処理の流れになる。なお，会社情報ファイルには企業についての基本情報が入力されているので，業種から企業を検索するといった処理の時は，3つのファイル全てを読み

込まなくても，256 KB しかないこのファイルへのアクセスだけで実行できる。

4．データ・ベース圧縮プログラム

磁気テープ形式の100メガ・バイトを越えるような大容量データ・ファイルを読み込み圧縮をする作業は，プログラミングと処理スピードの点で，どうしても汎用計算機で処理することになるが，この圧縮を行うための PL/I 言語によるプログラムのメイン・ルーチンは次のようになっている。

表5　データ圧縮プログラム例

```
010 CMPNEED  :  PROC OPTIONS(MAIN):
020  DCL INF FILE RECORD INPUT  ENV(FB RECSIZE(3000) BLKSIZE(15000));
030  DCL DAT FILE STREAM OUTPUT ENV(VB RECSIZE(1092) BLKSIZE (4368));
040  DCL IDX FILE RECORD OUTPUT ENV(FB RECSIZE  (71) BLKSIZE (7100));
050  DCL  1 IDATA,
060         2 IDA(273)DEC FIXED(9), 2 DUMMY   CHAR (1635);
070  DCL  1 ODATA,
080         2 ODA(273)CHAR(4);
090  DCL  1 XDATA,
100         2 XREC    CHA(2),       2 XDA(69)  CHAR(1);
110  DCL  1 HDATA,
120         2 C4      CHAR(4),      2 P        FIXED(2),
130         2 BT32    BIT(32),      2 BN31     BIN(31) FIXED,
140         2 BT(4)   BIT(2),
150         2 REC     FIXED(5),     2 LOD(273) FIXED(1),
160         2 PDREC   FIXED(15),    2 PXADR    FIXED(15),
170        (2 I, 2 N) BIN FIXED(15);
180 OPEN FILE(INF), FILE(DAT), FILE(IDX);
190  PDREC, PXADR=0  ;
200 /*-------------------------------------------*/
210 START  :  READ FILE(INF) INTO(IDATA);
220  REC=0 ; P=0 ;
230  DO I=1 TO 273  ;
240   P=P+1  ;
250   IF IDA(I)=-999999999 THEN DO ;
260     BT(P)='00' B ; LOD(I)=0 ;
270   END   ;
280    ELSE DO  ;
290     BN31=IDA(I) ; BT32=UNSPEC(BN31) ;
300     IF BN31>32767! BN31<-32768 THEN DO ;
310       BT(P)='11'B ; LOD(I)=4 ; REC=REC+4 ;
320     END  ;
330     ELSE IF BN31>127! BN31<-128 THEN DO ;
340      BT(P)='10'B ; LOD(I)=2 ;  REC=REC+2 ;
350     END  ;
360      ELSE DO  ;
370       BT(P)='01'B ; LOD(I)=1 ; REC=REC+1 ;
380      END  ;
390    END  ;
```

```
400     UNSREC(C4)=BT32 ;
410     ODA(I)=SUBSTR(C4,4,1)!! SUBSTR(C4,3,1)!!SUBSTR(C4,2,1)
420                          !! SUBSTR(C4,1,1) ;
430     IF P=4 THEN DO ;
440       N=N+1 ; P=0 ;
450       UNSPEC(XDA(N))=BT(1)!! BT(2)!! BT(3)!!BT(4) ;
460     END  ;
470    END  ;
480    BT(2), BT(3),BT(4)='00' B ;
490    UNSPEC(XDA(N+1))=BT(1)!! BT(2)!! BT(3)!! BT(4) ;
500    PUT FILE (DAT) EDIT
510      ((ODA(I) DO I=1 TO 273))(A(LOD(I))) SKIP ;
520    BN16=REC ; BT16=BN16 ; UNSPEC(XREC)=BT16 ;
530    XREC=SUBSTR(XREC,2,1)!! SUBSTR(XREC,1,1) ;
540    WRITE FILE(IDX) FROM(XDATA) ;
550    PDADR=PDADR+REC ;  PXADR=PXADR+71 ;
560    GOTO START ;
570   /*.....................................................*/
580  FINISH : CLOSE FILE(INF), FILE(DAT), FILE(IDX) ;
590    END CMPNEED ;
```

※ このプログラムは実際に使用したものではなく，圧縮方法の例示すためのものであり，273項目の Packed Decimal 5 bytes 数値を変換・圧縮し，実データ・ファイル（DAT），インデックス・ファイル（IDX）に書き込むと言う前提に立っている。なお，インデックス・ファイルの各レコードの先頭には，当該データの収録バイト数が2バイト整数で書き込まれている。

データ圧縮の中心部分をなす，Packed Decimal からパソコン・4バイト整数への変換は次の4つの処理（代入と変換）からなっており，特に複雑な処理でも，特殊な操作でもない[6]。

(1) Packed Decimal →31ビットのバイナリー数値　290行-1

(2) 31ビットのバイナリー数値→その内部表現の32ビット文字列　290行-2

(3) 32ビット文字列→それを内部表現とする4バイトの文字列　400行

(4) 4バイト文字列の順序を前後を逆に並び変える　410行

※(2)と(3)を一つの式，例えば，UNSPEC（C 4）= UNSPEC（BN 31）；としてもよい。

2バイト型，1バイト型の整数とは，この4バイト文字列の先頭から2バイトを取り出すなら2バイト整数，という具合になるので，これ以上の代入変換の操作は必要なく，出力のときに文字列出力バイト数を調節すればよいだけである。プログラムでは，260-2，310-2，340-2，370-2行で，LOD（I）に個々の実データ出力バイトを与え，それを510行の出力フォーマットで使用している。

実データ1項目ごとの入力様式を示すインデックス部分は，個々の項目の数値の大きさによって，2ビットの文字列に適当な値を与え（260-1，310-1，340-1，370-1行），これが4個まとまれば，その8ビットを内部表現とする1バイトの文字に変換し（430～460，480～490行），実データ1レコードを出力するときにインデックスも1レコード出力している（540行）。

以上がプログラムの基本構成である。今回実際に使用したプログラムの全文は，260行，約

200ステップ程度であるので，極めてコンパクトなプログラムと言えよう。

5．圧縮データ出力パソコン・プログラム

圧縮データをパソコンで読み込んで出力する操作は，出力したい企業，決算期のインデックス・データと基本データを読み込んだ上で，インデックスに基づいて基本データの0～4バイトを正しい数値に変換し出力するという流れになるが，8086系のCPUパソコンでのアセンブラ言語によるプログラムは表6に示すようになる。このプログラムは出力データをすべて4バイト整数の形式で出力するものであるが，表6の30～60行のような条件が与えられこのルーチンがコールされると，インデックス・データから1バイトを読み込み（150行），そこから2ビットを4回に別けて取り出し（180，190行），1回毎に取り出した2ビットが00_bなら欠損値－999999999にあたる$C4653601_h$の4バイトを出力し，01_bなら実データから1バイトを，10_bなら2バイトを読んだ後4バイト表現にし（Double Words）出力する，11_bなら4バイトを読み込みそのまま出力する，という処理になっている。

表6　圧縮データ読み込みパソコンプログラム

```
010 ;  ------------------------------------------------------------
020 ;          圧縮データ  →              ４バイト整数
030 ;                      DS:SI   =   実数データのアドレス
040 ;                      DS:BX   =   インデックス・データのアドレス
050 ;                      ES:DI   =   ４バイト整数出力アドレス
060 ;                      CX      =   連続変換するデータ項目数
070 ;            CPU 8086/80286/80386
080 ; ------------------------------------------------------------
090 MISL        =          03601H
100 MISH        =          0C465H
110
120 CNV4BYTE :
```

6）PL/I言語のバイナリー数値（整数）は，通常のプログラミングにおいては最高が31ビットであるが，その内部形式は，パソコンのC言語などの整数と同じで，2バイト（16ビット）と4バイト（32ビット）を単位としている。これには正負の判定用の1ビットが含まれているので，BINARY（1）～BINARY（15）は内部モードでは16ビット，BINARY（16）～BINARY（31）は32ビットである。この内部モードのコードをUNSPEC関数などで取り出す時，バイト単位の順序はパソコン様式とは前後逆となっている。なお，バイナリー数値をビット文字列にUNSPEC関数などで変換をすることなく直接代入すると，その値の絶対値（正の数値）がバイナリー宣言での指定ビット数だけ，前詰めでビット文字列に代入されることになる。つまり，次のようなプログラムで，40行で2つの変数（BN 16とBN 15）数値に代入された値が正の数値なら，50，60行の出力は全く同じで，'00001010' Bを出力するが，もし40行で－10が代入されているなら，50行の出力は'11110110' Bであり，2の補数で示される－10を表すが，60行は'00001010' Bを出力する。

```
010  DCL 1 TATA,
020        2 BN 16 BIN(16) FIXED, 2 BN 15 BIN(15) FIXED,
030        2 BT 16 BIT(16) ;
040     BN16, BN 15 = 10 ;
```

```
130             CLD
140             MOV     BP, CX
150 CNV4_0:     MOV     AL, [BX]        ;index 1 byte を AL に得る
160             INC     BX
170             MOV     CL, 4
180 CNV4_1:     SHL     AX, 1
190             SHL     AX, 1           ; AH に index 2 bit を得る
200             MOV     CH, AH
210             TEST    AH, 3
220             JZ      CNV4_5
230             JP      CNV4_2
240             TEST    AH, 1
250             JNZ     CNV4_3
260             LODSW                   ; 10b 処置
270             CWD
280             JMP     CNV4_6
290 CNV4_2      MOVSW                   ; 11b 処置
300             MOVSW
310             JMP      CNV4_7
320 CNV4_3:     LODSB                   ; 01b 処置
330             CBW
340             CWD
350             JMP     NV4_6
360 CNV4_5:     MOV     AX, MISL        ; 00b 処置
370             MOV     DX, MISH
380 CNV4_6      STOSW
390             MOV     AX, DX
400             STOSW
410 CNV4_7      MOV     AL, CH
420             DEC     BP
430             JZ      CNV4_8          ; 指定項目数の出力が終了したら
440             DEC     CL
450             JNZ     CNV4_1          ; 2 bits 単位での４回繰り返し
460             JMP     CNV4_0
470 CNV4_8:     RET
```

5-1. 圧縮データ出力用パソコン・ドライバー

圧縮データのパソコンでの出力は，表6で示したようなプログラムを作成することで，ユーザーが自由に行うことができるが，表6の操作を実行するには，前もって，(1) データ出力を必要とする企業情報ファイルをよみ，その企業についての，インデックスと実データの収録開始ポインタを得て，(2) インデックス・ファイルを読み込みながら，その企業の出力が必要な決算期のインデックスの取り込み，(3) 実データ・ファイルの必要箇所を取り込む，という操作が必要である。これを出力プログラムを作成するときに，いちいちプログラミングするのはかなり厄介な事である。そこでプログラミングを容易にするために，これらの操作をしてくれる，ツール・ソフトをドライバの形で登録できるようにしておくのが便利であろう。このような考えで，MS—DOS を OS とする8086系 CPU のパソコンなら共通して使用出来るドライバ・ソフト INTZZ を作成した。これは，ユーザーが希望する任意の割り込み番号に，この

圧縮データの読み込みルーチンを登録しておくもので，ユーザーは，MS—DOS のシステムコールと同じように，レジスタ AH に適当な値を与えて，この割り込み番号をコールすることで，圧縮データの基本的な出力を得られるようにしたものである。

詳細は省略するが，おおよそ次の機能がある。

割り込み番号：ユーザーが指定する任意の番号（0-255）（但し，OS によって許可されているもの）

(1) AH = 0 ： 圧縮データの3つのファイルのパス名を与え，ファイルをオープンし，初期化する

(2) AH = 1 ： 3つのファイルをクローズすることで終了する

(3) AH = 2 ： 出力したい企業コードを与えると，その企業情報（企業名，株式コード，上場情報等を出力する

(4) AH = 3 ： 指定企業の収録決算期情報を出力

(5) AH = 4 ： 指定企業指定決算期の入力データの全出力

(6) AH = 6 ： 指定企業の指定した決算期間の，指定した項目データを指定順に出力

(7) AH = 8 ： (6)と同じであるが，年度換算データを出力する

このドライバ・ソフトを使用すれば，圧縮データの出力プログラムの作成にあたって，ユーザーはこのデータの圧縮方式について，一切の配慮を払う必要がないので，比較的簡単にプログラミングすることができよう。

5-2．圧縮データのパソコンでのアクセス・スピード

このように作成した圧縮データと，出力用ドライバ・ソフトであるが，パソコンでのアクセスに実際どれほどの時間が必要であろうか。2つのパソコン機種でそのアクセス・タイムを計ってみた。実データ部分に入力されている263項目のデータについて，ランダム・サンプリングした100社について収録されている全決算期を出力した場合と，それらの年度換算を計算出力した場合などについて処理時間を計ると表7のようになった。

表7 圧縮データのパソコンでの処理時間（100社263項目全決算期の出力）

	B16HX モデル386		PC 9801 RX	
	全決算期	年度換算	全決算期	年度換算
ハード・ディスク使用	38.3	38.7	63.3	63.7
一部ラム・ディスク使用	35.0	35.3	59.0	59.0

※ ここでデータを出力するとは，4バイト整数の内部表現でメイン・メモリの指定アドレスにデータを書き込むことで，CRT に表示するとか，ファイルに出力したものではない。上段は3つのファイルをすべてハード・ディスクに置いた時で，下段は企業情報ファイルをラム・ディスクにおいた時である。B 16 HX モデル386は CPU が80386（32 bits），動作クロック16 MH_zで内蔵ハード・ディスク（40 MB）を使用，PC 9801 RX は80286（16 bits），12 MH_zで，ハード・ディスクは外付けの PCHD 040（40 MB）を使用した。なお，両機種とも通常の BIOS コマンドでは1秒以下の時間を計ることはできないので，3回テストして平均をとっている。

機種の性能，特にCPUのクロック数により，処理スピードはかなり異なるが，1社・全決算期データの読み込み・変換・出力が，約0.4～0.6秒であり，収録全社のデータをすべて読み込んで，比較・計算をするというような特別の処理をするような場合（収録全社の「売上高・営業収益（NEEDSデータ項目の90番）」を87年度について読み込み，高額から順にランキングづける計算したところ，PC 9801 RXでは4分48秒という結果となった）を別とすると，通常のデータ出力処理のスピードは，リアル・タイム処理と言える十分なスピードであろう。

おわりに

データ圧縮の技法は，図形，文書，バイナリー型のプログラム・ファイルなどに関して，様々な方法が開発されているが，NEEDSデータの様に，同一フォーマットで継続的に更新され，また研究者の利用も多いデータ・ベースに対しては，継続的な更新への対応を損なわない形で，それぞれのデータ・ベースに対応したデータ圧縮法が開発されてよいのではないかと考える。なお，この際，パソコンで圧縮データを読み込むためのプログラミングが複雑になったり，その処理スピードがパソコンの限られた能力のために実用に耐えられないものであっては無意味であるので，データ圧縮は，パソコンでの出力ソフトの作成の容易さも念頭において行う必要がある。

筆者がここで試みたデータ・ベースの圧縮法は，固定小数点形式の数値データがほとんどの特定のデータ・ベースを対象としたものであり，データ・ベース一般の圧縮法を説明したものではない。しかし，固定小数点形式のデータについては，この圧縮法はかなり汎用的に利用出来る方法ではないかと考えている。多くの研究機関で高価で有用なデータを購入したり，蓄積しながらその有効利用が図れていない例をよく耳にするが，これまでは，ともすればデータだけが与えられ（購入され）ながら，ユーザー・ニーズに見合った便利で有効な出力や解析のソフトがなかなか提供されてこないという問題が多くあった。そのような中で，このような試みが，データ・ベースの一層の活用の契機の1つとなれば幸いである。

付記

本稿脱稿後，大蔵省が上場企業の有価証券報告書を88年度決算以降についてパソコンで利用出来るCD-ROMに記憶された形で発売することが発表された（日本経済新聞・89年8月22日・夕刊）。ようやく待ち望んでいたものが発売されるという感がある。本稿で説明した研究・開発の動機には，CD-ROMのような形でデータ・ベースを継続・更新するのが現行の研究機関の力だけでは困難であるため，その障害を迂回して避けて通ることになっても，パソコンで大容量データ・ベースを直接アクセスすることを可能にしようとする思いもあったからである。公的な機関が，データの入力形式などを完全にオープンにしてこのようなデータを提供してくれれば，筆者としても「迂回」の必要が，少なくとも上場企業の有価証券報告書についてはなくなるわけである。88年度以前のデータも含めて，利用可能になることを期待したい。